ULTRA-LOW-LATENCY AUTOMATIC ENDOSCOPIC IMAGE ORIENTATION STABILISATION

Wiebe Van Ranst, Toon Goedemé and Joost Vennekens
EAVISE Research Group, KU Leuven
Jan Pieter De Nayerlaan 5, Sint-Katelijne-Waver, Belgium

ABSTRACT

Minimally invasive surgery often makes use of endoscopes with an angled lens. When using such an endoscope, the on-screen image tends to rotate wildly, which may confuse the surgeon. A low-impact way of solving this problem is by using computer vision techniques to track the rotation of the incoming footage and then counterrotating the image. Such an approach has already been proposed in the literature, but it has never been examined whether it is possible to integrate such an algorithm into a state-of-the-art Digital Operating Room environment, where minimal latency is required. In this paper, we compare three different ways of implementing a counterrotation algorithm on the hardware that is available in the NUCLeUS Digital Operating Room of the company eSaturnus.

KEYWORDS

Endoscopy, Image stabilisation, Ultra-low-latency

1. INTRODUCTION

Minimally invasive surgery is a type of surgery where small incisions are made in order to operate at a remote location in the patient's body. These incisions make room for the required instruments, as well as an endoscope which acts as the surgeon's eyes during the procedure. The endoscope is commonly equipped with an angled lens (typically set at 30 degrees, see figure 1), in order to enable the surgeon to get a broader view. Moreover, this also makes it possible to look behind objects by turning the endoscope, thus revealing perspectives which would be concealed with a normal endoscope. While this extra information provides a significant benefit to the surgeon, the angled lens is also a source of confusion and disorientation, since the act of rotating the endoscope also rotates the on-screen image (Breedvel, 1997). For certain kinds of procedures, this confusion can be avoided by placing a small amount of water into, e.g., the abdominal cavity in order to provide the surgeon with a reference "horizon". In situations where this is not possible, the

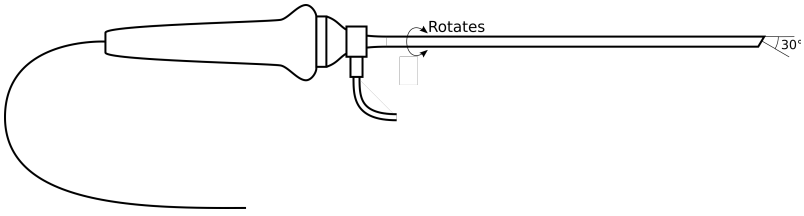

Figure 1. The endoscope lens rotates freely on the camera attachment

surgeon may choose to use an endoscope that allows its lens to be rotated independently of the endoscope itself (see figure 1). A downside of this solution, however, is that the movement required to turn only the lens while keeping the endoscope steady is quite cumbersome and often requires both hands of the operator, while also restricting camera movement.

A more promising approach is to solve this problem by postprocessing the video footage coming from the endoscope. Such postprocessing may consist of automatically counter-rotating the output of the endoscope, or of drawing a virtual "horizon" on screen, which informs the surgeon of the current rotation of the image.

In both cases, the core problem is that of determining the current rotation angle. Generally speaking, there are two ways of doing this: either by using some kind of hardware gravity sensor (Höller et al., 2009), or by using the images captured by the endoscope. While the use of a gravity sensor has the advantage of being drift free, an obvious disadvantage is that it requires a special endoscope equipped with such a sensor. In addition, gravity sensor endoscopes do not work if they are held above the patient, looking down.

By contrast, the image-based approach has the advantage that counter-rotation can be implemented entirely in software. Indeed, by tracking the translation between consecutive frames of the video feed, the orientation of the video can be automatically determined. Such an approach has already been investigated (Moll et al., 2009), but—to the best of our knowledge—it has never been implemented in a system used for live surgery. One of the main problems that need to be tackled in order to make this possible is that the video feed needs to be rendered at minimum latency. Indeed, for the hand-eye coordination of the surgeon, latency is a crucial parameter (Rayman et al., 2005). A recent study found that a latency above 130ms has detrimental effects on surgeon's performance (Kumcu et al., 2013).

The company eSATURNUS has developed a state-of-the-art digital operating room environment, called NUCLeUS^{TM}, which operates at extremely low latency: it streams 1080p60 video with a latency of <16ms, i.e., less than one frame. In this paper, we investigate how a video-based rotation tracking algorithm can be successfully integrated in this system. To do this, we have to tackle the unique challenge of combining high-framerate, high-resolution images with the requirement of ultra-low latency. We investigate the implementation of the algorithm on embedded hardware as a method for coping with these challenges.

This research has resulted in the development of a prototype, that can be seen in action at: http://www.youtube.com/watch?v=2tMk0u0yWc0.

The remainder of this paper is organized as follows. In section 0, we lay out the algorithm we used for our implementation, while section 0 explains how the algorithm is implemented on resource constrained hardware. Finally, section 0 discusses results in terms of performance and accuracy.

2. ROTATION COMPENSATION

The requirement for ultra-low latency forces us to consider a relatively simple approach to computing the rotation of the video. For this reason, we avoid feature matching techniques, such as SIFT (Lowe, 1999) or SURF (Bay et al., 2006), and have chosen to use a tracking algorithm instead.

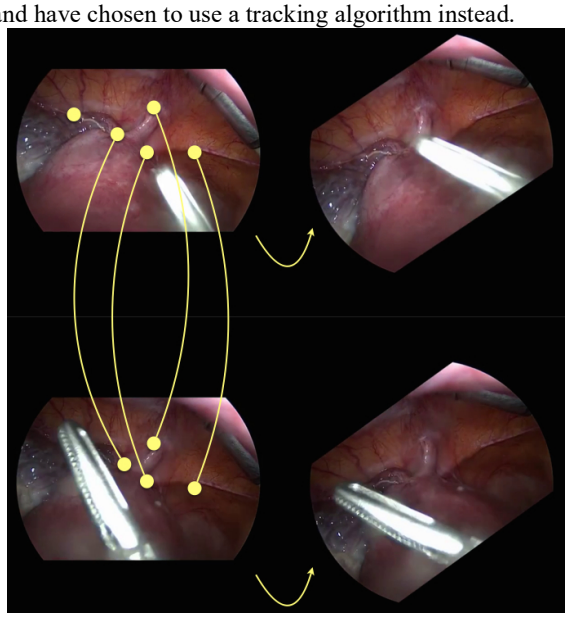

Figure 2. The tracking approach

The general idea of such a tracking approach is illustrated in figure 2. The left-hand side of this figure shows two consecutive frames of the incoming video. A number of distinctive points in the image (also called "corners") are extracted and these are then tracked across subsequent frames. From the movement of the corners from one frame to the next, the rotation angle between the two frames can be calculated. This angle can then be used to rotate the frame back to the reference orientation. The resulting "straight" frames are shown on the right-hand side of this figure.

To implement this approach, we have chosen to use a KLT tracker (Lucas and Kanade, 1981; Tomasi and Kanade, 1991). Our general approach is as follows. Processing starts when the operator enables the rotation compensation. This is done by pressing a button on the endoscope, while holding it level. At that point, the system takes the most recent frame of the stream and applies the corner detection algorithm. Once the corners have been extracted, the system can start tracking them through the next frames. However, over the course of the video, the system may lose track of some of the original corners. Once the number of tracked corners drops below a certain threshold, the corner detection algorithm is again applied and the process continues. We now discuss each of these components in more detail.

2.1 Detecting Corners

To detect corners in the endoscopy images, we use Harris corners (Harris and Stephens, 1988), which are known to produce good results for video footage in which there are small changes between consecutive frames (Schmid et al., 2000). We have also tried Shi and Tomasi's "Good features to track" (Shi and Tomasi, 1994), and found that they produce almost identical results on typical endoscopy footage (see figure 3). As suggested in the OpenCV manual, we track only corners that have a "quality level" of at least 0.01. On average, this results in about 170 corners being tracked.

2.2 Tracking

We use a pyramidal implementation of the KLT tracker as described in (Bouguet, 2001). The implementation makes use of a scale space pyramid built on top of the original image. Each new layer is constructed by

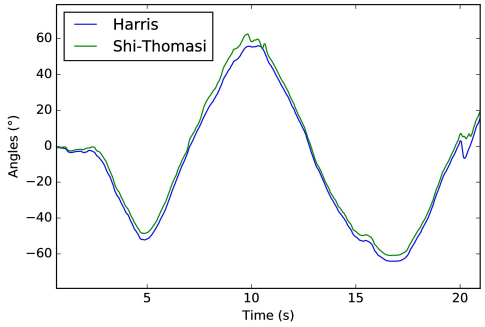

Figure 3. Harris vs. Shi-Tomasi

downscaling the previous layer by some constant factor. Feature tracking then commences at the top—i.e., lowest resolution—layer. These rough results are then gradually refined by descending in the pyramid.

2.3 Calculate Transformation

In this step, two point clouds—one from the previous frame and one from the current frame—are compared to each other in order to calculate the transformation between two frames. We calculate the homography between both clouds using OpenCV's RANSAC (Fischler and Bolles, 1981) implementation.

2.4 Interpolation of Estimated Angle

The Kalman filter assumes that the change in rotation angle is constant over time and so it predicts the angle α_t at time point t by the following linear model:

$$\begin{bmatrix} \alpha_{t+1} \\ \dot{\alpha}_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \delta_t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_t \\ \dot{\alpha}_t \end{bmatrix}$$

Here, δ_t is the time elapsed between time points t and t+1 (because the angle computation might have to skip frames, δ_t may not be constant).

After making the new prediction α_{t+1}, the Kalman model is then updated using a weighted average of α_{t+1} and the new measurement, i.e., the angle that was computed by the KLT tracker (the observation vector contains only the new angle). The weights depend on a process covariance matrix Q and a measurement covariance matrix R. We have empirically determined the values $\mathrm{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 77 \end{bmatrix} and\ \mathrm{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ for these matrices. The high value for velocity in the process covariance matrix indicates that, the predicted velocity is often off by a large margin, when the surgeon makes sudden movements.

When frames have to be dropped, the Kalman filter may start to drift away from the real rotation angle. This can happen in particular when there are sudden camera movements that violate the Kalman filter's assumption of a constant α_t. In such a case, the Kalman filter may have to perform a large correction when a new angle finally does arrive. This can give a jerky effect in the video. Therefore, we pass the resulting angle to a moving average filter which further smooths out the jerkiness. This filter keeps track of the last n samples (in our case, n=10) and is updated at a constant interval. Our implementation does this in a separate thread. The output of the moving average filter is used as the final result.

3. IMPLEMENTATION

The general architecture of the NUCLeUS Digital Operating Room is shown in figure 4. Video is captured by the source module, which transmits a direct, full HD feed to the receiver module, which decodes this feed and displays it on screen. There is also a secondary, lower-resolution SD feed, which can be used for other purposes, e.g., it can be streamed over the internet to medical staff follow the procedure from a remote location.

This architecture provides us with different options for implementing the tracking algorithm:

- REC: The computation of the angle happens entirely on the receiver module.
- R&S: The KLT tracker runs on the source module, while the Kalman and moving average filters run on the receiver.
- SRV: The angle computation happens on the separate compute server.

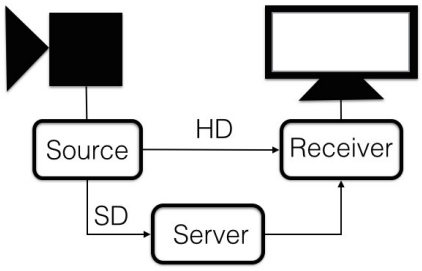

Figure 4. System architecture

We describe each option in more detail below.

3.1 Using only the Receiver Module (REC)

The receiver module contains a quad core processer and a Vivante GC2000 GPU, which supports the Embedded Profile (EP) of the OpenCL standard (Maghazeh et al., 2013). The REC approach uses this GPU to run the KLT tracker. When a new frame comes in, it is first downscaled to a resolution of 800x448px and then copied to GPU memory space. The GPU then calculates each level of the image pyramid of the frame, as well as its derivatives in both x and y directions, using the Scharr operator (Scharr, 2007). In all cases, a single work item is launched for every pixel in the output image.

Once the image pyramid is computed, the actual tracking starts. We launch one workitem for each corner in the previous frame. On a typical desktop GPU, this would use only a fraction of the available compute units. However, since the Vivante GC2000 GPU only has four compute units, this parallelization strategy is able to keep the available hardware occupied. Inside the kernel, the displacement of each feature is calculated using the KL tracking algorithm.

Finally, we transfer the updated locations of the corners back to the CPU and use OpenCV to calculate the transformation between the current and previous point cloud. We then apply the Kalman and moving average filter.

3.2 Using both the Receiver and the Source Module (R&S)

This variant splits the computation over the receiver and source module: the source calculates the rotation angle and the receiver applies the Kalman and moving average filter. One advantage of using the source module to calculate the rotation angle is that we can make use of the already downscaled secondary video stream that is available on this module, while in the REC approach, the full HD stream still has to be downscaled on the receiver. Because the source module does not have a GPU, this implementation uses the OpenCV CPU version of the KLT algorithm instead of our own OpenCL EP implementation. The resulting angle is transmitted to the receiver module as part of the header of the image that is sent, so no additional communication delay is incurred.

3.3 Compute Server (SRV)

This approach performs the tracking on the separate desktop machine that gets the secondary SD feed. The resulting angle is sent to the receiver module, which uses it to rotate its direct HD feed and then renders the results it on-screen.

Because the angle and HD image arrive through different paths, they need not be synchronized: the receiver always uses the most recent angle it has received to rotate the most recent image. Because of this, the angle computation cannot increase the latency with which the video feed is rendered. If the angle computation is too slow, this will have as its only effect that the rotation will use an angle that is slightly off. The Kalman filter on the receiver module can compensate for this effect, while the additional moving average filter avoids jerkiness upon receiving a delayed angle.

If the compute server is powerful enough to keep up with the framerate, however, then no interpolation of the angle is necessary. In this case, the Kalman filter just smooths out sudden movements, at the expense of causing the rotation angle to lag slightly behind.

4. RESULTS

In this section we evaluate our different implementations in terms of both performance and accuracy.

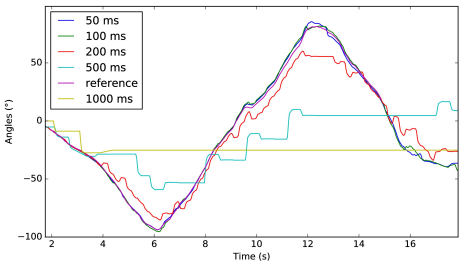

Figure 5. Angles at different update rates

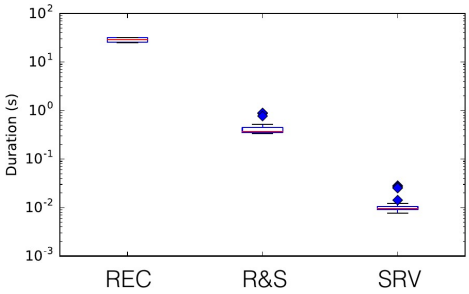

Figure 6. Runtime comparison

If the computation of the rotation angle cannot be performed fast enough to keep up with the framerate of the video, our approach is to drop frames from the angle computation. In this way, we avoid increasing the latency with which the video is rendered, but this obviously comes at the cost of lowering the accuracy of the computed angle. Figure 5 examines the effect of increasing the interval between two angle computations on the accuracy of the angle. The baseline corresponds to computing an angle for each frame of the 30fps video. We can therefore achieve the baseline by computing each angle within 33ms. As the time needed for each frame increases, we have to drop more frames and the accuracy deteriorates. As figure 6 shows, we can go as low as 5fps and still have a reasonable approximation of the baseline.

Figure 6 shows the performance of our algorithm on the different system configurations discussed in section 0, measured over 40 trials.

The REC approach is infeasible, as it takes an average of almost 28s per frame. The Vivante GPU that is available on this module is therefore clearly not powerful enough to perform the required computation.

Shifting the computation of the rotation angle to the source module (R&S) provides a significant improvement. This is due to the fact that the source module's CPU (an Intel atom Z530) is more powerful than the processors that are available on the receiver, as well as the fact that we avoid having to downscale the video twice. However, the average computation time of 0.43s means that we could still only compute 2 angles per second. As we can see from figure 5, this is an unacceptably poor approximation of the real rotation angle. Moreover, the worst execution time for this approach was substantially slower (0.89s) than the average. Therefore, this approach has to be ruled out.

We can conclude that the approach with a separate compute server (Intel Xeon E5-2630 in our tests) is most feasible in this case. While the downside of this approach is obvious—i.e., the need for additional hardware—the gain in runtime outweighs the cost for this latency-sensitive application. Indeed, with an average runtime of 11ms and a worst runtime of 28ms, this solution is able to keep track with the 30fps framerate of the video. Taking also into account the network transfer of the low-resolution image feed to this compute server and the communication of the angle from the server to the receiver, the latency with which the computed angle arrives at the receiver module is approximately between 50 and 100ms. This slight lag in rotation angle is not noticeable in practice.

As discussed before, the video itself is rendered from the direct HD feed, so angle computation does not affect its latency at all. However, the GPU that is available on the receiver module of NUCLeUS cannot actually counterrotate the HD video in 60fps. One solution is to work with 1280x720@30fps video instead. The other is to simply render the incoming HD feed as is, but to impose a "horizon" on it that informs the surgeon of the current rotation. We have implemented both approaches.

5. CONCLUSION

To cope with the disorientation arising from the use of angled endoscopes, we have developed an orientation stabilisation method, based on the KLT tracker (Lucas and Kanade, 1981; Tomasi and Kanade, 1991), and examined how to integrate this into the NUCLeUS Digital Operating Room. This state-of-the-art system aims at providing an ultra-low-latency solution, which is important since research suggests that latency is a key parameter for surgeon's performance. The aim of this paper is therefore to investigate methods of integrating orientation stabilisation into this system without adversely affecting this latency.

We have developed two embedded implementations of the KLT tracker: an OpenCL EP implementation running on the GPU in the receiver module and an implementation running mainly on the CPU in the source module. Our experimental results demonstrate that neither of these implementations is fast enough to allow a reasonable approximation of the rotation angle. However, the IP-based architecture of NUCLeUS also allows a third option, namely that of running the bulk of the computation on a separate desktop machine. This options allows the latency with which the video is rendered to remain unchanged, while the rotation angle that is computed never lags more than a few frames behind.

ACKNOWLEDGMENT

This work is supported and partially carried out at eSATURNUS nv. This work was supported by IWT Flanders via the Hippocrates project.

REFERENCES

Bay, H., Tuytelaars, T., Van Gool, L., 2006. Surf: Speeded up robust features, in: Computer vision–ECCV 2006. Springer, pp. 404–417.

Bouguet, J.-Y., 2001. Pyramidal implementation of the affine Lucas Kanade feature tracker. Intel Corp. 5, 1–10.

Fischler, M.A., Bolles, R.C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24, 381–395.

Harris, C., Stephens, M., 1988. A combined corner and edge detector., in: Alvey Vision Conference. Citeseer, p. 50.

Höller, K., Penne, J., Schneider, A., Jahn, J., Boronat, J.G., Wittenberg, T., Feu\s sner, H., Hornegger, J., 2009. Endoscopic orientation correction, in: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2009. Springer, pp. 459–466.

Kumcu, A., Elprama, S., Vermeulen, L., Duysburgh, P., Platisa, L., Van Nieuwenhove, Y., Van De Winkel, N., Jacobs, A., Van Looy, J., Philips, W., 2013. Effect of video latency on performance and subjective experience in laparoscopic surgery, in: 15th Medical Image Perception Society Conference, Abstracts. pp. 59–59.

Lowe, D.G., 1999. Object recognition from local scale-invariant features, in: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on. IEEE, pp. 1150–1157.

Lucas, B.D., Kanade, T., 1981. An Iterative Image Registration Technique with an Application to Stereo Vision, in: IJCAI. pp. 674–679.

Maghazeh, A., Bordoloi, U.D., Eles, P., Peng, Z., 2013. General purpose computing on low-power embedded GPUs: Has it come of age?, in: Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII).

Moll, M., Koninckx, T., Van Gool, L., Koninckx, P., 2009. Unrotating images in laparoscopy with an application for 30° laparoscopes, in: 4th European Conference of the International Federation for Medical and Biological Engineering.

Rayman, R., Primak, S., Patel, R., Moallem, M., Morady, R., Tavakoli, M., Subotic, V., Galbraith, N., Wynsberghe, A. van, Croome, K., 2005. Effects of Latency on Telesurgery: An Experimental Study, in: Medical Image Computing and Computer-Assisted Intervention (MICCAI), Lecture Notes in Computer Science.

Scharr, H., 2007. Optimal filters for extended optical flow. Springer.

Schmid, C., Mohr, R., Bauckhage, C., 2000. Evaluation of Interest Point Detectors. Int. J. Comput. Vis. 37.

Shi, J., Tomasi, C., 1994. Good features to track, in: Computer Vision and Pattern Recognition (CVPR). IEEE, pp. 593–600.

Tomasi, C., Kanade, T., 1991. Detection and Tracking of Point Features. Carnegie Mellon University.